Wege und Irrwege der Informatik-Ausbildung

David Lorge Parnas,Ph.D, Dr.h.c., Dr.h.c., FRSC,P.Eng.

Director of the Software Engineering Programme
Computing and Software, Faculty of Engineering
McMaster University, Hamilton ON Canada L8S 4K1

Abstract

When the first Computer Science/Informatik programmes were developed, few people anticipated the role that computers play in our society today. Computers and software are a critical part of our infrastructure and we are as dependent on them as we are on roads, pipelines, and sewers. In the future we will be even more dependent.

Our failure to understand that computers would become utilities led us to follow a questionable path in educating software developers. We educate them as we educate philosophers, mathematicians and scientists, not the way that we educate engineers and doctors. We also failed to recognise that workers should possess basic qualifications before being allowed to work on this critical infrastructure. Consequently, many unqualified people cooperate to produce unreliable critical products.

This talk will propose alternative approaches to the education of software specialists, approaches that will prepare students for the immense responsibility of developing our critical "infostructure" (information infrastructure).

Scientists and Engineers: Contrasts

- "Science is the study of what is. Engineering is the creation of what never was?" (Theodor von Karman)
- Scientists add to the world's knowledge. Engineers add to the world's set of products and tools.
- Scientists produce knowledge that help Engineers and often function as Engineers. Engineers produce scientific devices and often function as scientists.
- Scientists can be narrow and specialised. Engineers are expected to have a broad base of knowledge and are responsible for the whole product including being sure that it is fit for its intended use.
- In Science we see specialisation early in education. In engineering and other professions, specialisation comes later.

Regulated vs. Unregulated Professions

Scientists' work is checked by professionals. Engineer's work is tested by the public.

As a scientist, I study facts, produce reports and papers. Programs or devices are prototypes for evaluation and study, not products. The papers are read by other scientists and (some) by engineers who can check my work and understand its conditions of validity.

Engineers often work for amateurs who can not judge their competence.

Licensing is introduced to assure the public of the competence of an identified set of professionals.

For each profession, a "core body of knowledge" is identified. A license is issued only to those who have that knowledge and are aware of the relevant laws and regulations.

Some licensed professionals are more skilled than others; some lose their license for incompetence and/or negligence. The majority are "OK"

The system is imperfect but far better than no regulation at all.

Science Education vs. Professional Education

Science is usually an unregulated profession with a "liberal" education; students have a great deal of choice and can study what interests them.

Engineering, Medicine and other regulated professions use a different model. *Professional Education* is designed to prepare people to work as a licensed professional.

Professional Education is rigid and forces the student to learn the "core body of knowledge" in order to get a degree.

In Computer Science/Informatik we have taken the Science Model. This was inherited from our "parent" field, Mathematics.

For CS, we have not identified a "core body of knowledge" that is known by all CS graduates. If I meet CS graduates, I do not know what they know or what words they will understand in the way that I understand them.

Software Engineering: A Meaningless Term?

A term becomes meaningless if it can be applied to anything or nothing.

The following are among the possible meanings:

- A new discipline of engineering: design of software intensive artifacts
- Building software
- Managing software projects
- Using SQL
- Subset of Computer Science, relevant to software design
- Working in teams, Negotiation
- Cost estimation
- Configuration control and management,
- Applied Psychology, problem solving, etc.
- Manufacturing instructions, the more the better
-

"Software Engineering" is used as a German word, but I am not sure what it means in either language.

In Canada we have already had one lawsuit about the meaning of this term.

Traditional Engineering - Purpose and History

Engineers: three meanings (different roots).

- People who build things for others (e.g. roads)
- People who used ingenuity
- People who know about engines

The first two should coincide. The third is outmoded.

Frequent failures caused by unprepared practitioners.

Accidents endangered the public.

Licensing required to assure **minimum** preparation.

Licensing authorities, examine, remind, and discipline. You can lose your license to practice if judged negligent by a group of your peers.

Disciplines were introduced within Engineering because of limited education time. We cannot teach all we know within 4 or 5 years.

Engineering core (about 1.5 years) is valid for all disciplines. Only two or three technical options available - 4th year.

Software Engineering - History

Early 60s:

- Programming done by scientists and mathematicians.
- Engineers not interested, no physical products.
- Where's the "Engineering" in SE? (1986)
- Recognition that writing software is not science.
- Shouldn't we do it (teach, construct products) as the Engineers do it?

1967: 1968 - The famous NATO conferences.

- Did not involve many professional engineers.
- Stressed cost, schedules, planning etc. rather than science/math based discipline of design.

1970 - present

- ***Not*** doing it as the engineers do it.
- Software Science confused with Engineering.
- Many viewpoints, more management than engineering.
- No licensing, No accreditation, No broadly accepted core body of knowledge.
- No focus on science/math based design.
- No improvement in quality of practice.

Engineers Ask: Is "Software Engineering" Engineering?

What's the *real* question?

- Is what today's self-anointed "software engineers" do, engineering?
- Is what is talked about at the International Conferences on Software Engineering engineering?
- Is what is in the IEEE Transactions on S.E. engineering?
- Is what is taught at the CMU SEI engineering?
- Is most SE research engineering?
- Is the use of engineering principles, sound science, and mathematics to develop products that affect the safety and well being of the public, engineering?
- Is there a need for professionals who have been taught the appropriate science, mathematics, and engineering disciplines for building software?

My answers are: NO, NO, NO, NO, NO, YES, YES.

Software Engineers Are Not Just Good Programmers

"Software Engineer" often a euphemism for "programmer".

Many assume that all a software engineer does is write code.

An Engineer is responsible for producing *products* that are "***fit for use***"; software is not used in isolation.

Making a product is fit for use, requires understanding the environment in which it is used.

"Software Engineers" should know many things outside computer science.

Software engineering programmes must stress software design, but must also equip students with knowledge about other areas of engineering.

- They must understand be able to analyse the interaction of the software with the rest of the system.
- They must know when to call for help from other engineers and be able to speak their "language".

However, they must be very good program designers and experienced programmers!

Software Engineers are not Necessarily Managers

Management requires the ability to get something done without knowing exactly what it is.

Engineers are responsible for knowing the properties of what they have built.

The best Engineering Managers were Engineers first and many could still practice Engineering.

Our inability to design software well, Our inability to document software properly, makes software projects very hard to manage.

The solution is to teach design and documentation, not *just* project management.

Why is this an issue for FIFF or CPSR?

As human beings we have many and responsibilities and are concerned about human rights, privacy, peace and many other issues.

Our common bond is our understanding of computers and software.

We must advise the public on issues where our professional knowledge gives us insight that is not shared by the general public when that insight is essential to their safety and well-being.

Today, the safety and well-being of the public depends on software.

We must:

- Warn the public that software is not always trustworthy.
- Warn that many software "experts" do not have the appropriate expertise.
- Demand, *and assist with,* the development of product standards.
- Demand, *and assist with,* the development of standards for personnel,
- Insist on a licensing and an enforcement mechanism
- Insist on accreditation of educational programmes for people being licensed.

We must not oppose use of technology, but see that it is used properly.

Would Licensing and Accreditation Violate Basic Rights?

This is a typically American interpretation of freedom. It looks at the rights of the coder.

Users, and the rest of the public have rights too!

- They have the right to know whether or not products meet standards.
- They have the right to know whether or not developers meet standards.
- They have the right to know whether educational programmes meet standards.

The standards must be "minimal" and objectively defendable.

No programme should be "closed" simply accurately labelled.

The right to *knowingly* hire an unlicensed professional for private tasks must remain.

Licensing authorities must be responsible to the public, not the profession.

Decisions must be appealable and defensible.

Licensed professionals must be able to supervise unlicensed professionals if they take responsibility for the work.

Is it too Early to License Software Professionals?

ACM has rejected licensing because “we do not yet know how to produce perfect programs.

- Perfection is not a prerequisite for licensing
- We do not yet have perfect buildings, roads, or aircraft.
- The perfect is the enemy of the good.

Today our fellow citizens are heavily dependant on our profession and that dependency will only grow.

Our citizens have a right to know who they can trust and for what.

Is one Kind of License Enough?

Not all software developers have to be Engineers!

The Engineering core is not required for many applications.

Other knowledge is more useful for applications such as (non-real-time) operating systems, word processors, banking, ...

In Canada all Engineers have the same license, but each is clearly obligated not to practice outside of the discipline(s) in which they are competent.

Engineers do not always know when they are outside of their area of competence.

My preference is to have two licenses, Professional Software Engineer, and Professional Software Developer. Each would have a clearly specified area of practice.

- Engineers: Applications requiring an understanding of physical systems
- Developers: Applications requiring no physics but a deeper knowledge of information processing.

What about Computer Scientists/Informatiker?

Scientists remain essential in every area.

- Physicists develop information and materials for Electrical Engineers
- Chemists are obviously essential for developing new materials to be produced in factories developed by Chemical Engineers.
- Scientists often have deep, specialised, knowledge needed by Engineers.
- Scientists make new discoveries; Engineers apply them.

Every student should have the opportunity to choose between Professional Education and traditional Science Education.

Differences In Topic Coverage.

Many topics in computer science are interesting and challenging, but have not yet found practical application. Examples are:

- Denotational semantics of programming languages,
- Neural computation,
- Many parts of “artificial intelligence”
- Some aspects of computability and automata theory.

Extensive discussion of those topics ***must*** be included in CS programmes.

SE programme should explain why they are not (yet) important.

In contrast, a graduate of an SE programme should understand aspects of communication, control theory, and interface design that are rarely seen in CS programmes.

Every educational programme is a compromise.

Every CS course has some interest to SEs and the reverse should be true.

Similar to the situation with Physics and Electrical Engineering

Differences in Course "Style" And Content

CS students relatively patient and willing to explore topics just because they are interesting.

Engineering students become impatient if they are not shown how to apply what they are learning.

From engineering students, "That course is theoretical" is criticism; From CS students - praise.

There will be many topics that should be covered in both programmes but we may have to give quite different courses.

Example: Mathematical Logic:

For SE: emphasise the use of logic to describe properties of systems and properties of states. checking specifications and programs for completeness and consistency. Students should use theorem provers.

CS students would learn the differences between various kinds of logics, and to discuss issues such as generalised decision procedures and the meaning of non-denoting terms for which the SE course would have little time. Students may build theorem provers. Remember that in one programme we are teaching students to apply well-established techniques, in the other we should teach them how to add new techniques.

Respecting Career Realities

Most Engineers are not people who develop new technology; they apply proven technology.

The education should reflect this: more focus on using tools than developing tools.

Few SE graduates will end up designing either operating systems or compilers.

Understanding Compilers and Operating Systems is fundamental for a CS graduate. They are just another application for an SE.

Data base systems are important tools for SE graduates. They are an interesting area of study for Computer Scientists.

Inside the Computer vs. Complete System

The most popular SE examples look naïve to traditional Engineers.

Engineers must be sure that their products will do the job. Satisfying a specification prepared by a non-Engineer is not enough. "*Fit for use*" is the slogan.

It is impossible to do the job without understanding of the environment and control mechanisms.

Building the right thing is a lot more than building the thing right!

The Famous "Cruise Control" Problem

Cruise control "solutions" have appeared in the literature for more than a decade.

They assume a binary/trinary decision, accelerate, decelerate, or maintain throttle level.

This is naïve because:

- Throttle can be varied continuously, not a trinary value.
- You cannot tell if the solution will be satisfactory without an understanding of engine and vehicle characteristics.
- This is a control problem (with piece-wise discontinuous functions describing the driver).
- CS treats it as a small finite state machine.
- As CS views it, the problem is essentially trivial. ***Any*** method will work with a small FSM.

Most CS graduates would not have any idea how to start on such a problem? Is this acceptable for Engineers specialising in software?

Nuclear Plant Fuel Rod Replacement

Problem: Given a design for a reactor in which the fuel "burning" rate is not uniform, new fuel rods should not be near each other, and power generation capability should remain relatively constant, devise an optimal replacement policy in the form of a program that specifies rod rotation policy.

This sounds like an ideal problem for a "Software Engineer", but

- Today's SEs know little physics and cannot produce the necessary mathematical models of the situation.
- Today's SEs do not get a course in optimisation methods.
- Today's SEs, CS graduates, are likely to propose an *ad hoc* (heuristic) solution that is far from optimal.

Who should do this problem? Physicists or Engineers who cannot program? Computer Scientists without the proper physics and optimisation background, or Software Engineers?

Processing of Communications Data

Computer Scientists tend to live in a world in which there are large data sets that are noise-free. The problems come from the size and complexity of the data set, not the noise.

Communications software must deal with systems that handle relatively small amounts of noisy data.

The two groups rarely understand each other.

Since so much software is developed for communications networks with large amounts of data, a Software Engineer must understand both views.

Software Engineers involved in data communications should understand coding, information theory, etc., topics that are rarely covered in CS curricula. They must understand data base and other techniques for dealing with large amounts of data, a topic that Engineers rarely study.

Predicting Performance

Traditional Engineers do not determine the load bearing capacity of a structure by "cut and try". They use mathematics to compute capacity.

There are relevant results in operations research that can provide useful performance predictions, given accurate usage data.

Most Computer Science graduates do not learn this; it is not CS, it comes from other areas.

Software Engineers should be exposed to this area. CS students need not learn it.

- Design of Parallel/Distributed Computer Systems and Computations
- Computer Networks and Computer Security
- Software in Communications Systems

Guiding Principles that are Often Forgotten.

(1) Integrate Theory and Practice
- No theory without showing how to use it.
- No "practical" technique without showing that it is fundamentally sound.
- Use what you teach in projects and assignments or don't teach it.

(2) Teach technology in labs, lasting material in lectures
- No languages or systems named in course
- No "fads", "buzzwords", Schlagwörter
- Lecture material could have been useful 20 years earlier; should be useful 30 years from now.
- Both technology and fundamental principles are important but students must learn the difference.

Should All IT Workers have a University Education?

My opinion: Many University courses in Computer Science are not appropriate for either Universities or Technical Universities.

They teach current technology, opinions that are not scientifically based.

Much of what is taught is out of date before graduation or soon thereafter.

Students do not know the fundamentals that are required for professional growth in a dynamic field.

Students do not learn the thinking habits needed to evaluate new ideas.

These students are “technicians” not Scientists or Engineers.

Every Engineering field has its technicians who work with Engineers.

We need them too.

The public needs them.

A Summary: University Programmes

Type	Core Knowledge	Type of Programme	Applications
Engineer: Software	Core Engineering Basic software design, algorithms, Discrete Mathematics,	“Rigid” engineering style education focusing on fundamentals.	Computer Systems interacting with physical systems. Use of well understood technology.
Professional Software Developer	Discrete Math, algorithms, advanced software design, tool development.	“Rigid” engineering style education focusing on fundamentals.	Design of information systems well understood technology.
Computer Scientist, specialising in software	No clear definition. Intended for future researcher.	Classical flexible university education	Research, new tools, new algorithms, notations, methods.

Non-University Programmes

Type	Core Knowledge	Type of Programme	Applications
Professional Programmer	Programming languages and Support systems.	“College” training in specific applications and application tools.	Technician Support for Software Engineers and Professional Software Developers
Software Maintainer/ Installer	Commonly available system software	Technician level training in specific products and basic ideas	Support staff for users.
Specialised Technician	Single product or small set of products.	“Commercial training course	Support/help-line etc.

The important step in planning a software engineering curriculum is to define what you mean and make it clear which of these alternatives you want to prepare.

European vs. North American Standards

My experience in Europe is limited but:

- I found the students better prepared.
- I found them more interested in learning principles.
- I found them a bit more capable.
- I found them more mature.
- I found them more serious.
- I found the Professors (most not all) less interested in what was going on in practice than they are where I live.
- The U.S. system does not fit European norms and is not an improvement.

What is our responsibility as responsible professionals.

- Make the public aware of the problems of an unregulated profession.
- Work for the establishment of Professional Standards and Licensure.
- Work for the establishment of curriculum accreditation standards.

Make sure that what is done is in the best interest of the public, not the profession or the industry. They are not always the same.